STA141A Final Project

Manish Rathor and Victor Bulyga

9/10/2023

Abstract

In this project, we trained an ARIMA model to predict future Google closing stock prices using data from 2018-2023. We first imported the dataset and performed an exploratory analysis to understand its contents. Once we developed an understanding of the data's components, we cleaned and transformed the data so it was suitable for visualization. After that, we created relevant visualizations to analyze trends and the overall shape of the data. The visualizations, along with our prior knowledge of stock market and time-series data, confirmed that we needed to train an ARIMA model for accurate forecasting. Before training the model, we had to transform the data again, because the relevant time-series functions in R cannot be applied to data frames. After performing the necessary transformations, we created new visualizations to further understand underlying trends within the data. Then, we were able to train the model. We split the original dataset into a training and test set, trained the model on the training set, and created a forecast with the model. We then compared the test data to the forecast through relevant visualizations, which gave us a rough idea of the model's accuracy. After that, we tested the model's performance by first checking to ensure that it satisfied relevant assumptions, and then evaluating its accuracy through its RMSE. Once we were confident that our model was accurate, we decided to use it in a simulation. Through this simulation, we determined that the model would generate profit, although due to Google's abnormally high stock value, it did not generate as much profit as naively buying shares and never selling. Finally, we included a brief discussion regarding the applications and importance of this type of analysis.

Introduction

In the Big Data Era, harnessing data and statistical models has allowed businesses to create accurate predictions for all sorts of problems. Using these models has increased efficiency and allowed firms to guide their decision-making processes with concrete insights.

One instance of this is seen in the financial industry. Investment firms, which generate profits through the buying and selling of financial commodities, play a risky game. They hedge their bets on the success or failure of the commodities they purchase, and can experience financial failure if their gambles don't pay off. To minimize the risk of their bets, these companies rely on data-driven predictions. They train machine learning models to predict which commodities will succeed and fail, allowing them to buy and sell with minimal risk.

In our project, we will train a model that accurately predicts future stock values. We will specifically analyze Google stock data ranging from July 30, 2018 to July 28, 2023. The data we will use contains the daily opening, high, low, closing, and adjusted closing prices, as well as the daily volume of the stock.

- Opening Price: The price of a stock when the market opens
- High Price: The highest price of a stock in a given period
- Low Price: The lowest price of a stock in a given period

- Closing Price: The price of a stock when the market closes
- Adjusted Closing Price: The closing price after adjustments for splits and dividend distributions
- Volume: The number of shares traded in a given period

To simplify this analysis, we will solely focus on the daily closing price of the stock. This is an industry standard, as most firms either use closing price of adjusted closing price data to train their models.

Our first step will be to handle and pre-process the data. To ensure that we can implement the necessary functions and accurately train the model, we will convert our data to the right types and remove any corrupt/incomplete data.

After handling and pre-processing, we will visualize the data. Creating visualizations will help us decide which methodology to use while selecting and training our model.

Once we've visualized the model, we will select the appropriate statistical model and train it.

Finally, after selecting, training, and implementing the model, we will analyze its performance and discuss the implications of our findings.

Data Wrangling/Preparation

We start by importing the data. As seen below, the data is imported from a .csv file and is stored as a data.frame. As stated above, the data contains the opening, high, low, closing, and adjusted closing stock prices for Google. We can also see that the observations begin on July 30, 2018.

```
# Loading in the dataset
google.df <- read.csv("GOOG.csv")
kable(head(google.df), "pipe")
```

Name	Date	Open	High	Low	Close	Adj.Close	Volume
GOOG	7/30/18	61.4005	61.7458	60.57350	60.9870	60.9870	36998000
GOOG	7/31/18	61.0005	61.3794	60.28000	60.8630	60.8630	32894000
GOOG	8/1/18	61.4000	61.6735	60.51050	61.0005	61.0005	31344000
GOOG	8/2/18	60.2950	61.4940	60.23950	61.3075	61.3075	30626000
GOOG	8/3/18	61.4810	61.5000	60.75300	61.1855	61.1855	21792000
GOOG	8/6/18	61.2500	61.3044	60.78985	61.2385	61.2385	21634000

```
class(google.df)
```

```
## [1] "data.frame"
```

We use this function to find the final observation of the data, which occurs on July 28, 2023.

```
# Finding the final date in the data
kable(tail(google.df, n = 1), "pipe")
```

	Name	Date	Open	High	Low	Close	Adj.Close	Volume
1258	GOOG	7/28/23	130.97	134.07	130.92	133.01	133.01	26959800

Our first step in pre-processing the data is converting the Date values to date type. As you can see above, they are currently stored as characters, which prevents us from visualizing and modeling the data. The function below converts the values from character to date.

```
# Converting the data to date type
google.df$Date <- as.Date(google.df$Date, format = "%m/%d/%y")
kable(head(google.df), "pipe")
```

Name	Date	Open	High	Low	Close	Adj.Close	Volume
GOOG	2018-07-30	61.4005	61.7458	60.57350	60.9870	60.9870	36998000
GOOG	2018-07-31	61.0005	61.3794	60.28000	60.8630	60.8630	32894000
GOOG	2018-08-01	61.4000	61.6735	60.51050	61.0005	61.0005	31344000
GOOG	2018-08-02	60.2950	61.4940	60.23950	61.3075	61.3075	30626000
GOOG	2018-08-03	61.4810	61.5000	60.75300	61.1855	61.1855	21792000
GOOG	2018-08-06	61.2500	61.3044	60.78985	61.2385	61.2385	21634000

Now that we have converted the Date to date type, we will convert the Volume to type double. While it is already stored as a numeric, meaning that it can be used in our statistical analysis, we will convert it to double to maintain consistency with the rest of the data.

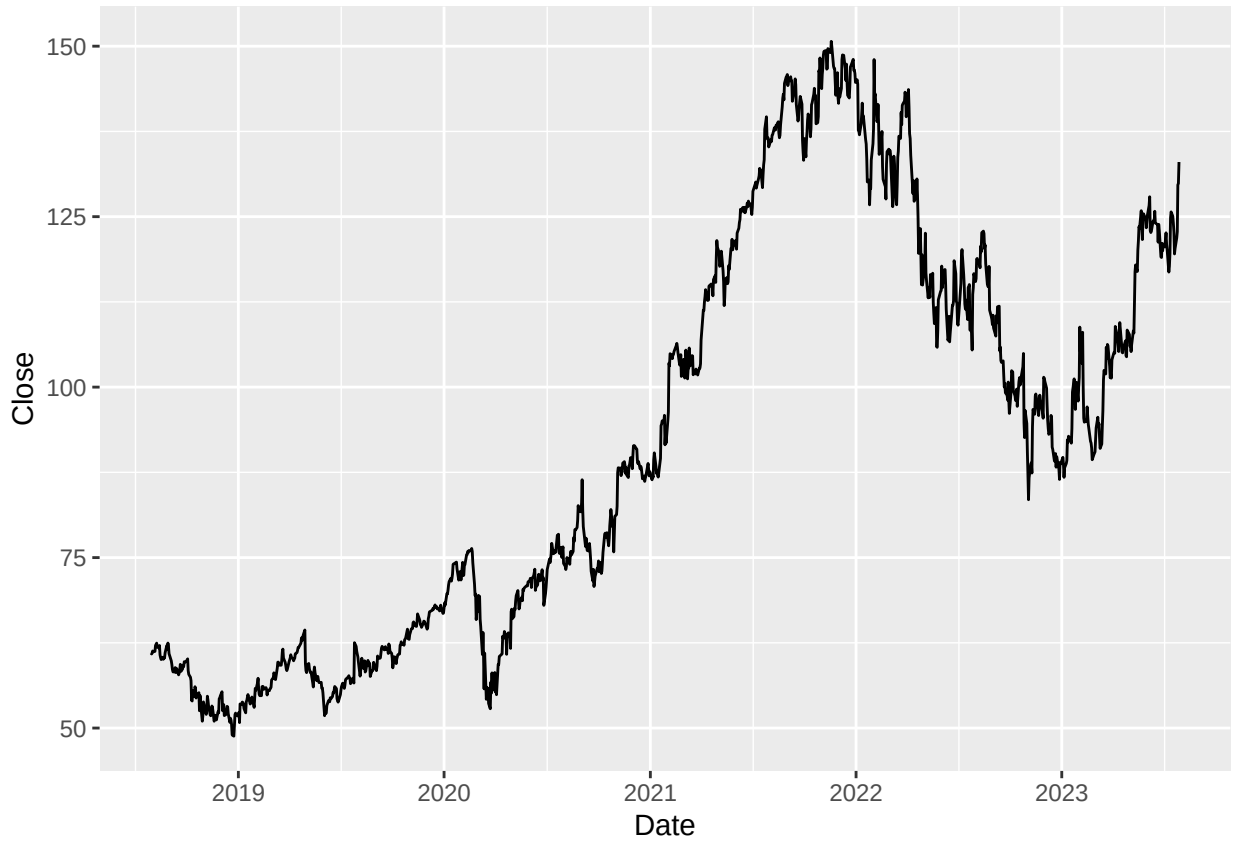
```
# Converting the Volume to numeric type
google.df <- google.df %>%
    mutate(Volume = as.numeric(Volume)) %>%
    na.omit(.) %>%
    subset(select = c(Date, Open, High, Low, Close, Volume))
kable(head(google.df), "pipe")
```

Date	Open	High	Low	Close	Volume
2018-07-30	61.4005	61.7458	60.57350	60.9870	36998000
2018-07-31	61.0005	61.3794	60.28000	60.8630	32894000
2018-08-01	61.4000	61.6735	60.51050	61.0005	31344000
2018-08-02	60.2950	61.4940	60.23950	61.3075	30626000
2018-08-03	61.4810	61.5000	60.75300	61.1855	21792000
2018-08-06	61.2500	61.3044	60.78985	61.2385	21634000

Data Visualization

Now that we have converted the Date to date type, and ensured that the rest of the values are doubles, we can visualize our data. As we said before, we will be focusing on the closing price, so we will use a line graph to visualize the closing price across all five years. To do this, we will use the ggplot2 package, which allows us to create aesthetic visualizations with clean, concise code.

```
# Visualizing closing price data
google.df %>%
    ggplot(aes(x = Date, y = Close)) + geom_line()
```



Based on the visualization, we can see that while the data follows a general trend, there are fluctuations throughout. We can conclude from this visualization, and from our understanding of stock market data that a time-series analysis will be required. Time-series models are able to account for the fluctuations (known as seasonality) and create accurate predictions.

A Brief Introduction to Time-Series Analysis

Most of the data we are used to is made up of observations that contain the same features (predictors) and target (output). When developing regression models, we train the models to predict future target values using the features. For example, if we are given a dataset that contains the age of 20 women, and their respective muscle mass, we can train a linear regression model to predict muscle mass for women at different ages. The model will do this by analyzing the relationship between age (feature) and muscle mass (target). It will then generate a mathematical formula that can accept a hypothetical age value and output a hypothetical muscle mass.

Time-Series data differs in the sense that there are no features. Instead, the target data is collected repeatedly over a certain period of time. Stock market data is an example of this. The price of a stock is recorded on a daily basis and is stored in a table. To model and predict this data accurately, we have to take a different approach. Rather than using features to predict the target, we have to use previous target values to predict future target values. This will be demonstrated below.

More Data Wrangling/Preparation

In R, the relevant functions to implement a time-series analysis can be found in the quantmod, xts, and forecast libraries However, for the functions within the libraries to work, we need to transform the data into a new type.

Because we have already tidied our data previously, the only required step is to transform the data from class data.frame to class xts. Converting the data to this format will allow the functions from the previously desribed libraries to work.

```
# Converting dataframe to ts class
google.ts <- as.xts(google.df)
kable(head(google.ts), "pipe")
```

Open	High	Low	Close	Volume
61.4005	61.7458	60.57350	60.9870	36998000
61.0005	61.3794	60.28000	60.8630	32894000
61.4000	61.6735	60.51050	61.0005	31344000
60.2950	61.4940	60.23950	61.3075	30626000
61.4810	61.5000	60.75300	61.1855	21792000
61.2500	61.3044	60.78985	61.2385	21634000

```
class(google.ts)
```

```
## [1] "xts" "zoo"
```

As we can see, the data is now in class xts.

More Data Visualization

Now that we have transformed the data, it is a good idea to visualize it. Rather than using the ggplot2 package, we will use a function from the quantmod package, which is designed to visualize stock market data.

```
# Visualizing the data with a new chart
chartSeries(google.ts, name = "Google Price 2018-2023")
```

As we can see, the chart not only contains the line graph of the stock data, but it also contains a visualization of the volume traded. Much of this is unnecessary for our analysis, but it is a good way to visualize a stock's overall performance. We can also see that the chart looks the same as the one produced using the ggplot2 package, confirming that none of the values in the data have been changed.

There is one more important visualization we would like to look at: a decomposition graph. This graph will visualize the different trends within the data more effectively than our current visualizations. However to do this, we need to simplify the data.

We will simplify the data by creating a new xts that only contains the monthly closing price for the stock. By transforming the data so that there is only one closing price per month, we can effectively generate a decomposition graph.

```
# Filtering the ts object to only include monthly closing prices
google.ts.close <- Cl(to.monthly(google.ts))
kable(head(google.ts.close), "pipe")
```

google.ts.Close
60.8630
60.9095
59.6735
53.8385
54.7215
51.7805

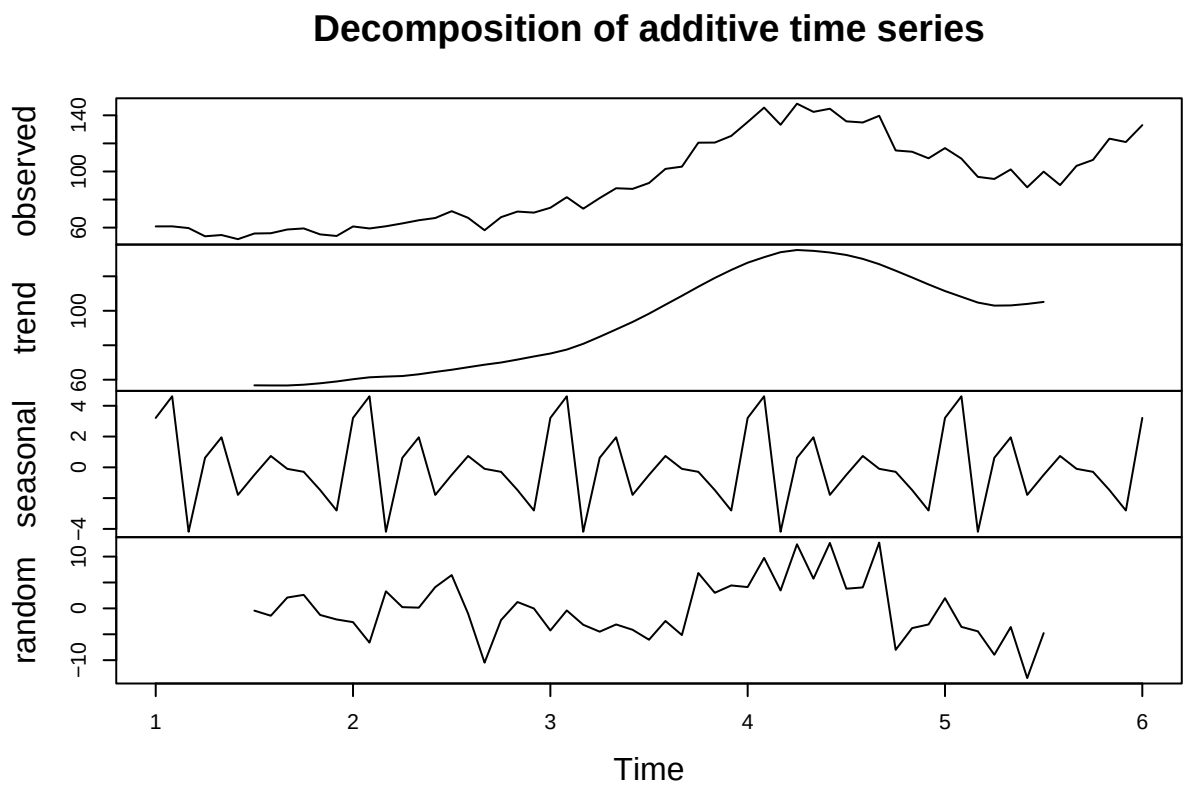
```
class(google.ts.close)
```

```
## [1] "xts" "zoo"
```

As we can see, the new object contains a single closing price for each month, and remains in the class xts.

To create the decomposition graph, we will apply the decompose() function to the new xts object we've created.

```
# Decomposing the data to visualize relevant trends
dc <- decompose(as.ts(google.ts.close))
plot(dc)
```



As we can see, the decomposition graph gives us four different charts: Observed, Trend, Seasonal, Random. Time is written a 1-6, but represents 2018-2023.

- Observed: This is the original plot of the data. The observed data is shown below.

```
dc$x
```

```
##         Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
## 1  60.8630  60.9095  59.6735  53.8385  54.7215  51.7805  55.8185  55.9960
## 2  60.8340  59.4050  60.9500  63.0055  65.2480  66.8510  71.7115  66.9665
## 3  74.1480  81.7090  73.4800  81.0505  88.0370  87.5940  91.7870 101.8430
## 4 135.2210 145.4620 133.2655 148.2705 142.4520 144.6795 135.6985 134.8910
```

```
## 5 116.6400 109.1500  96.1500  94.6600 101.4500  88.7300  99.8700  90.3000
## 6 133.0100
##        Sep      Oct      Nov      Dec
## 1  58.6655  59.4240  55.1815  54.0455
## 2  58.1405  67.4330  71.4460  70.6805
## 3 103.4315 120.5060 120.5780 125.3160
## 4 139.6495 114.9665 114.0390 109.3725
## 5 104.0000 108.2200 123.3700 120.9700
## 6
```

- Trend: This visualizes long term movements in the data. We can see that it generally reflects the shape of the Observed graph, but does not contain the minor fluctuations. The trend data is shown below.

```
dc$trend
```

```
##         Jan       Feb       Mar       Apr       May       Jun       Jul
## 1        NA        NA        NA        NA        NA        NA  56.74192
## 2  60.28092  61.40023  61.83546  62.14729  63.15869  64.52950  65.77737
## 3  75.20285  77.49252  80.83283  84.93133  89.18988  93.51352  98.33471
## 4 127.89731 131.10396 133.99004 135.26831 134.76504 133.82827 132.38975
## 5 111.45690 108.10608 104.76273 102.99623 103.10392 103.97594 105.14125
## 6        NA
##         Aug       Sep       Oct       Nov       Dec
## 1  56.67802  56.66852  57.10367  57.92423  58.99077
## 2  67.26146  68.71287  69.98683  71.68825  73.50208
## 3 103.53579 108.68323 113.97513 119.04325 123.68910
## 4 130.10254 127.04306 123.26281 119.32063 115.28098
## 5        NA        NA        NA        NA        NA
## 6
```

- Seasonal: This visualizes repetitive fluctuation in the data. As we can see, the closing price is highest at the beginning of the year, and then steeply drops to its lowest right after. This indicates to firms that they should buy Google stock a month or two after the year begins, and sell right at the beginning of the next year. The seasonal data is shown below.

```
dc$seasonal
```

```
##           Jan         Feb         Mar         Apr         May         Jun
## 1  3.20975500  4.61430480 -4.18538745  0.61933351  1.95086992 -1.78967970
## 2  3.20975500  4.61430480 -4.18538745  0.61933351  1.95086992 -1.78967970
## 3  3.20975500  4.61430480 -4.18538745  0.61933351  1.95086992 -1.78967970
## 4  3.20975500  4.61430480 -4.18538745  0.61933351  1.95086992 -1.78967970
## 5  3.20975500  4.61430480 -4.18538745  0.61933351  1.95086992 -1.78967970
## 6  3.20975500
##           Jul         Aug         Sep         Oct         Nov         Dec
## 1 -0.49139675  0.73817433 -0.09666846 -0.29123451 -1.47446193 -2.80360877
## 2 -0.49139675  0.73817433 -0.09666846 -0.29123451 -1.47446193 -2.80360877
## 3 -0.49139675  0.73817433 -0.09666846 -0.29123451 -1.47446193 -2.80360877
## 4 -0.49139675  0.73817433 -0.09666846 -0.29123451 -1.47446193 -2.80360877
## 5 -0.49139675  0.73817433 -0.09666846 -0.29123451 -1.47446193 -2.80360877
## 6
```

- Random: This visualizes random fluctuations in the data not accounted for by the trend and/or seasonality. This is also known as white noise. Because much of the data was collected during the COVID-19 pandemic, there is quite a bit of white noise. It is important to note that larger amounts of white noise will lead to less accurate models, something to consider during the analysis. The random data is shown below.

```
dc$random
```

```
##            Jan          Feb          Mar          Apr          May          Jun
## 1           NA           NA           NA           NA           NA           NA
## 2  -2.65667179  -6.60953518   3.29992995   0.23887578   0.13844358   4.11117795
## 3  -4.26460654  -0.39782618  -3.16744372  -4.50016876  -3.10374292  -4.12984055
## 4   4.11392513   9.74374236   3.46084799  12.38284599   5.73608342  12.64091211
## 5   1.97334654  -3.57038768  -4.42734089  -8.95555968  -3.60479075 -13.45625618
## 6           NA
##            Jul          Aug          Sep          Oct          Nov          Dec
## 1  -0.43201896  -1.42019721   2.09364858   2.61156776  -1.26826836  -2.14166019
## 2   6.42552412  -1.03313346 -10.47570746  -2.26259974   1.23221101  -0.01797873
## 3  -6.05630983  -2.43096508  -5.15505904   6.82210543   3.00921439   4.43050627
## 4   3.80014792   4.05028908  12.70311125  -8.00508011  -3.80716370  -3.10487402
## 5  -4.77985158           NA           NA           NA           NA           NA
## 6
```

Modeling/Forecasting

Now that we have appropriately wrangled/pre-processed our data, and have also visualized it effectively, it is time to select and train our model.

The specific model we will use is the Auto-Regressive Integrated Moving Average (ARIMA) model. The ARIMA model stems from the idea that current target values depend on previous target values, as well as previous changes and errors. This model is able to account for this through three parameters: P, D, and Q. P represents the number of autoregressive terms in the model, D represents the number of times the data is differenced, and Q represents the number of moving average terms in the model.

The ARIMA model is advantageous for a few reasons. Firstly, it can handle a wide range of univariate time series data. Also, it can account for trends, volatility, and seasonality. The ARIMA model is also easy to use, as it doesn't require many parameters or assumptions. Finally, because it is developed using statistical methods/theory, it generally produces accurate forecasts.

It is also important to note that the ARIMA model is one of the most widely used models in financial analysis (specifically for predicting stock performance). Based on the advantages described above, the characteristics of our data (univariate with seasonality and trend), and the fact that investment firms use the model often, we believe that it is an appropriate fit.

To train the model, we need to create a training and test dataset. We will do this by splitting our current data into two new datasets. Specifically, we will select the final 100 observations to be the test data, and keep the remaining data as the training data.

```
# Splitting the dataset for training and test sets
n <- 100
train <- head(Cl(google.ts), length(Cl(google.ts)) - n)
test <- tail(Cl(google.ts), n)
```

Now that we have created our training and test data, we can train the ARIMA model. The auto.arima() function creates the model, while the forecast() function will forecast future observations. We set h = n so that the forecast generates 100 future observations.

The auto.arima() function automatically selects the optimal model by using a variation of the Hyndman-Khandakar algorithm. The algorithm uses unit root tests and minimizes the AICc and MLE to generate the best parameters. This means that we do not have to manually select parameters and run diagnostic tests. Instead, the function will do the work for us and return the best model.

We will also train two different models. One will account for seasonality, and one won't. If both models return the same values, we can conclude that seasonality isn't a significant part of the data, and we can therefore use the non-seasonal model for our prediction.

```
# Training a non-seasonal ARIMA model
model <- auto.arima(train, seasonal = F)
forecast <- forecast(model, h = n)
model
```

```
## Series: train
## ARIMA(0,1,1)
##
## Coefficients:
##           ma1
##       -0.0637
## s.e.   0.0299
##
## sigma^2 = 3.475:  log likelihood = -2361.84
## AIC=4727.68   AICc=4727.69   BIC=4737.78
```

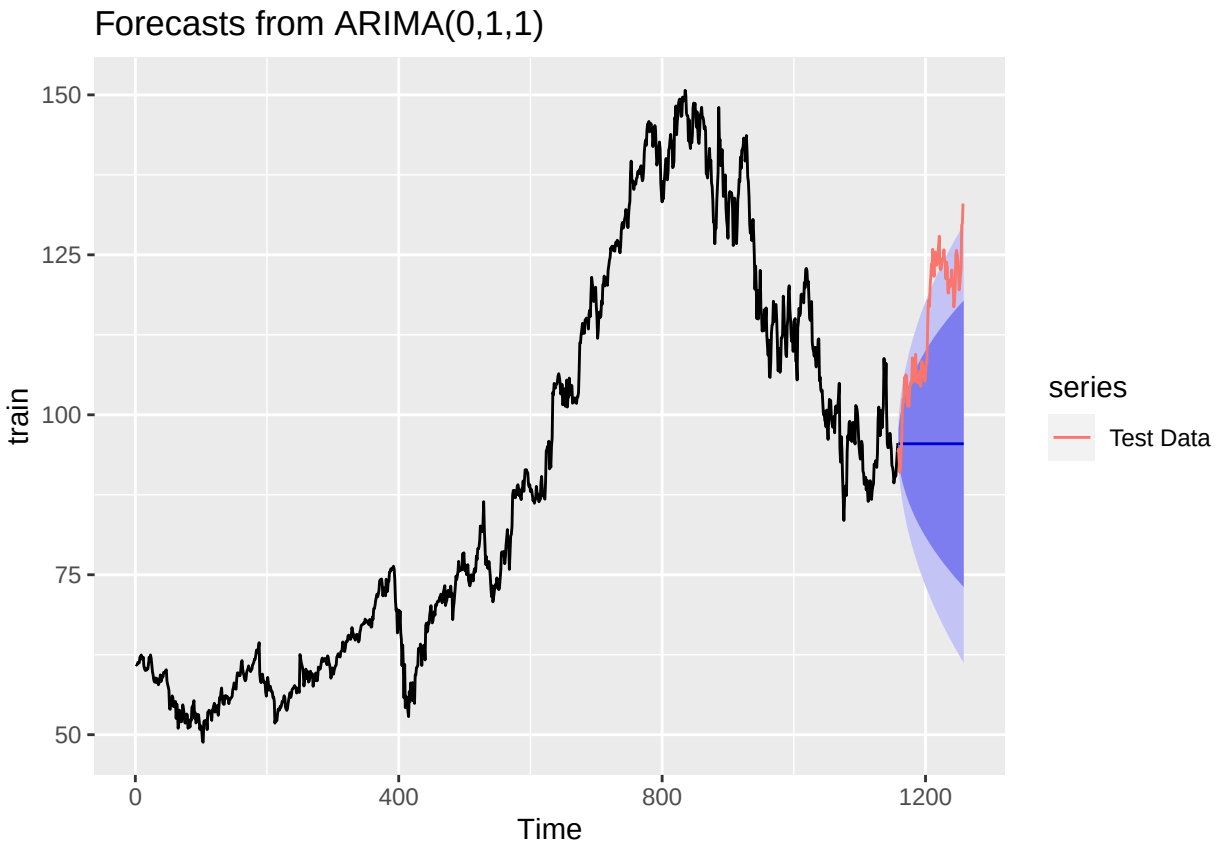
```
# Training the seasonal ARIMA model
model.s <- auto.arima(train)
forecast.s <- forecast(model.s, h = n)
model.s
```

```
## Series: train
## ARIMA(0,1,1)
##
## Coefficients:
##           ma1
##       -0.0637
## s.e.   0.0299
##
## sigma^2 = 3.475:  log likelihood = -2361.84
## AIC=4727.68   AICc=4727.69   BIC=4737.78
```

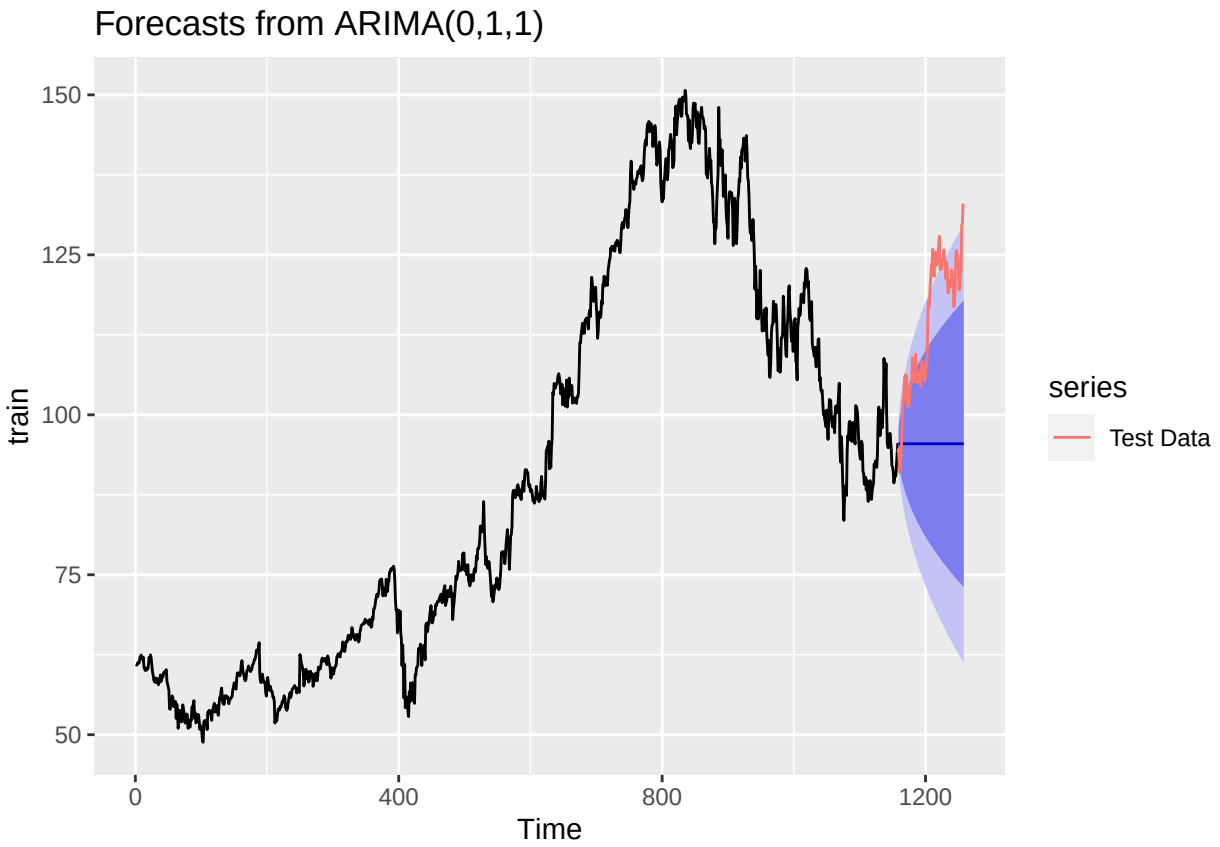
As we can see, the seasonal and non-seasonal models generate the same values. This means that seasonality is not significant, and we can use the non-seasonal model for our prediction.

We can also visualize the models. Because both models are the same, their visualizations will also be the same. However, we will generate charts for both to maintain consistency.

```
# Visualizing the non-seasonal model
autoplot(forecast) + autolayer(ts(test, start = length(train)), series = "Test Data")
```




```
# Visualizing the seasonal model
autoplot(forecast.s) + autolayer(ts(test, start = length(train)), series = "Test Data")
```



Based on the visualization, we can see that the model offers a good, but not great, forecast of the data. We can draw this conclusion from the confidence interval visualization on the right side of the graph. In this visualization, the thin dark blue line represents a 100% confidence interval, the dark blue arc represents a 95% confidence interval, and the light blue arc represents an 80% confidence interval. As we can see, the test data falls within the light blue arc, meaning that it falls within the 80% confidence interval. Therefore, the forecast is able to somewhat capture the test data values, but there is room for improvment.

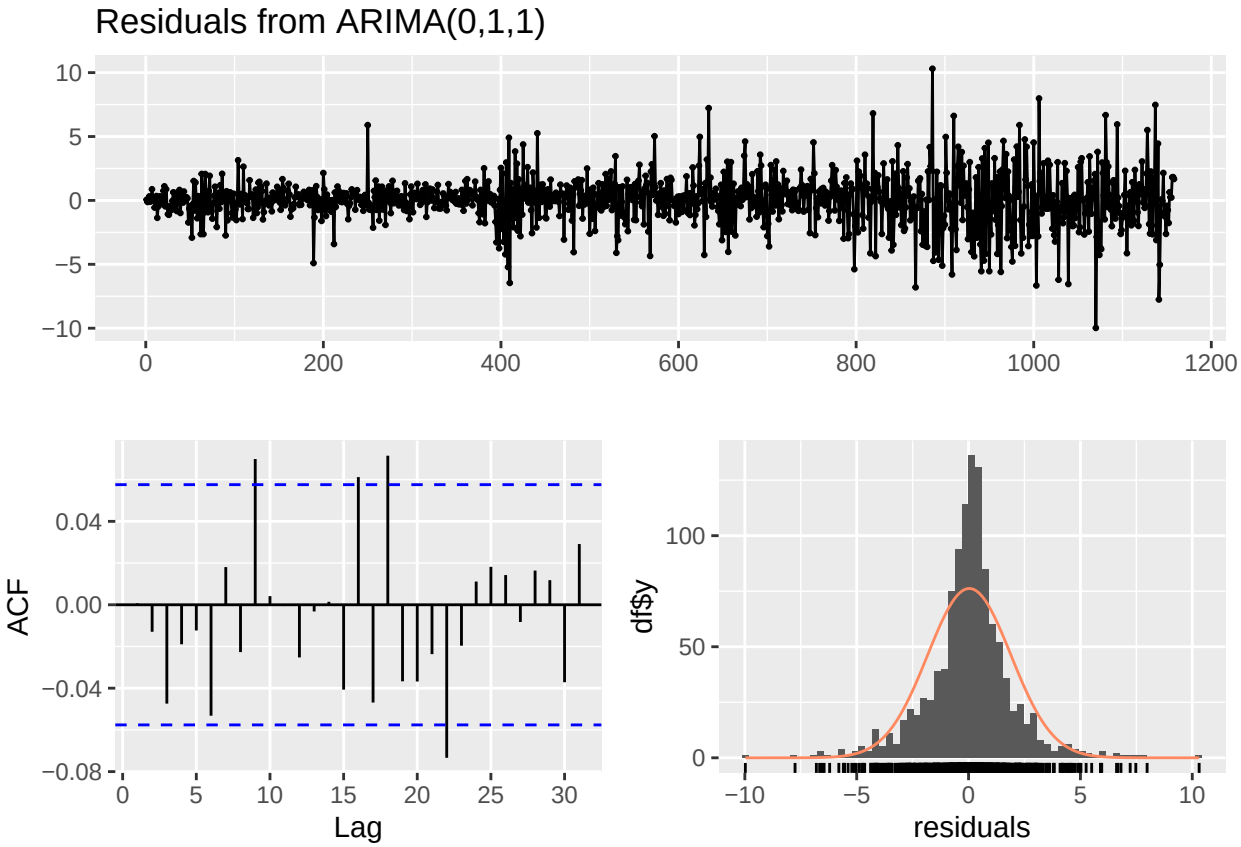
Results

Now that we have trained our model and forecasted our data, we can evaluate the performance.

The first step in this process is to check if the residuals meet the residual assumptions. The assumptions are listed below

- Residuals must be normally distributed
- Residuals must have constant variance
- Residuals must not be autocorrelated

```
# Checking the residuals of the model
checkresiduals(forecast)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)
## Q* = 13.414, df = 9, p-value = 0.1447
##
## Model df: 1.   Total lags used: 10
```

As we can see from the histogram, the residuals are normally distributed. Meanwhile, the residual plot shows that the variance is constant. Finally, to check for autocorrelation, we use the ACF plot and the Ljung-Box Test. With the ACF plot, if the lines extend beyond the upper or lower bound, there may be autocorrelation. With the Ljung-Box test, if the p-value is less than 0.05, there may be autocorrelation. However, in the case that the ACF plot and the Ljung-Box test produce different results, we will use the result from the Ljung-Box test.As we can see, while the ACF plot suggests autocorrelation, the p-value from the Ljung-Box test suggests the opposite. Because the two tests produce different results, we will use the Ljung-Box test result, and conclude that there is no autocorrelation.

Finally, we can check the accuracy of our model. We will use the accuracy() function to do this. However, the function generates a variety of statistics, so we will choose to focus on the Root Mean Squared Error (RMSE). The RMSE is the standard deviation of the residual, essentially measuring the variance of the residuals. Intuitively, we want a lower RMSE value.

```
# Checking the accuracy of the model
accuracy(forecast)
```

```
##                      ME      RMSE       MAE       MPE      MAPE      MASE
```

```
## Training set 0.03185866 1.862604 1.274606 0.0196275 1.418771 0.9996604
##                       ACF1
## Training set 0.0006686167
```

The guidelines for determining a "good" RMSE are dependent on the data being forecasted. Datasets with larger numbers should have larger RMSE values, while datasets with smaller numbers should have smaller RMSE values. Based on the numbers in the stock market data, it is reasonable to conclude that an RMSE value of 1.863 is very good. This means that the model is very accurate.

Another way to check a model's accuracy is through the normalized RMSE value. The formula for a normalized RMSE is RMSE/(max value - min value). For a normalized RMSE to be considered "good", it should fall between 0 and 1.

```
# Normalized RMSE calculations
RMSE <- 1.862604
maxClose <- max(google.df$Close)
minClose <- min(google.df$Close)
nRMSE <- RMSE/(maxClose - minClose)
nRMSE
```

```
## [1] 0.0182791
```

As we can see, our RMSE is very good. A value this close to zero means that the model can be considered accurate.

Simulation

Now that we have determined that our model is accurate, we want to test it in a real-world scenario. To demonstrate this, we will create a simulation in which a trader will use the model to guide their buying/selling process. The trader will begin trading on July 30, 2018, with $1,000 and zero shares. Every thirty days, the trader will buy or sell based on the model's predictions and certain parameters we set. This process will repeat until we reach the present date.

```
# Given the history of a stock, create a prediction of that stock's
# possible values
stockForecast <- function(stock.history) {
    train <- head(stock.history, nrow(stock.history) - n)

    # Training a seasonal ARIMA model
    model <- auto.arima(train)
    forecast <- forecast(model, h = n)

    return(forecast)
}

# Store information about simulated stock trader
trader.info <- data.frame(Liquid.Funds = 1000, Shares.Owned = 0)

tradeDecision <- function(stock.history, trader.info, next.choice) {
    price <- as.numeric(tail(stock.history, 1))
    funds <- trader.info$Liquid.Funds
    owned <- trader.info$Shares.Owned
```

```
    can.buy <- funds%/%price
    forecast <- stockForecast(stock.history)

    # first determine if trader is waiting to buy or waiting to sell
    if (owned == 0) {
        # if waiting to buy, wait until the stock doesn't have the
        # risk of dropping significantly in value by the time of the
        # next decision.
        if (as.numeric(forecast$lower[next.choice, 2]) > 0.7 * price) {
            choice <- "buy"
        } else {
            choice <- "hold"
        }
    } else {
        # if waiting to sell, hold until the stock doesn't show
        # promise for increasing significantly by the time of the
        # next decision.
        if (as.numeric(forecast$upper[next.choice, 1]) < 1.01 * price) {
            choice <- "sell"
        } else {
            choice <- "hold"
        }
    }

    # based on the current choice, make the changes to the trader's
    # portfolio
    trader.info <- switch(choice, buy = data.frame(Liquid.Funds = round(funds -
        can.buy * price, 2), Shares.Owned = owned + can.buy), sell = data.frame(Liquid.Funds = round(fu
        owned * price, 2), Shares.Owned = 0), hold = trader.info)
    return(trader.info)
}

trader.info <- tradeDecision(Op(google.ts), trader.info, 30)
```

```
trader.info <- data.frame(Liquid.Funds = 1000, Shares.Owned = 0)
trader.history <- trader.info
next.choice <- 30
n <- 100

# This for-loop simulates a trader making decisions each month by
# updating the trader's portfolio with the decision function. All of
# the decisions of the trader are recorded.
for (i in seq(2 * n, nrow(google.ts), next.choice)) {
    partial.history <- head(google.ts, i)

    trader.info <- tradeDecision(Op(partial.history), trader.info, next.choice)
    trader.history <- rbind(trader.history, trader.info[1, ])
}

kable(tail(trader.history), "pipe", align = "c")
```

	Liquid.Funds	Shares.Owned
32	67.47	17
33	67.47	17
34	67.47	17
35	67.47	17
36	2183.80	0
37	2183.80	0

```
17 * as.numeric(tail(Op(google.ts), 1))
```

```
## [1] 2226.49
```

Using our model, the trader would've had a final portfolio value of $2,226.49. This means that they would've experienced a $1,226.49 profit. However, if the buyer had simply bought in 2018 and never sold, they would've had a final portfolio value of $2,261.17, resulting in a $1,261.17 profit. Despite this, it doesn't necessarily mean that the model is bad. Instead, this situation is more reflective of the fact that Google's stock grows at an abnormally fast and consistent rate, without ever experiencing a significant drop in value. If the model is applied to most other stocks, which don't perform as well as Google, it is reasonable to infer that it would produce positive results.

Conclusion

As we can see from the forecasting results and the trading simulation, time-series modeling is a powerful way to successfully trade on the stock market. Applying this technique to the financial industry will allow businesses to improve their profits and minimize risk.

Beyond this model, there are also be other ways to predict stock performance. Using company information as features may yield accurate performance predictions. Also, using an algorithmic stock selection technique to find companies that will perform well is a viable idea.

Overall, the use of statistical modeling in the financial industry is an innovative way to improve on outdated trading methods.